Programação Orientada a Objetos
Turmas B e C

Introdução a Padrões de Projeto
(*Design Patterns*)
Aula Teórica

Nádia Félix
nadia.felix@ufg.br

Dirson S. Campos
dirson_campos@ufg.br

(24/02/2022)

Programação Orientada a Objetos

Definições

Padrão

"O padrão é uma descrição do problema e essência de sua solução, onde pode ser reutilizada em diversos casos. O padrão não é uma especificação detalhada, pode-se pensar como uma descrição de conhecimento e experiência acumulados." (SOMMERVILLE, 2003)

Programação Orientada a Objetos

Definições

- Em 1995, os profissionais: Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, escreveram e lançaram o livro chamado **"Design Patterns: Elements of Reusable Object-Oriented Software"**, que mostra os detalhes dos 23 Design Patterns. Por essa realização, os profissionais foram batizados com o nome "Gangue dos Quatro" (Gang of Four ou GoF).
  - O livro foi traduzido para o português com o nome Padrões de projetos – Soluções Reutilizáveis de Software Orientado a Objetos
- Os objetivos dos padrões de projeto, são tornar os componentes reutilizáveis, que facilitam a padronização e que permitam agilidade para as soluções de problemas recorrentes no desenvolvimento do sistema.

Programação Orientada a Objetos

Motivação

- Projetar software OO reusável e de boa qualidade é difícil
- Mistura de específico + genérico
- Impossível acertar da primeira vez
- Projetistas experientes usam soluções com as quais já trabalharam no passado

Outros Tipos de Padrões:

- Padrões de Análise:
  - Descrevem grupos de conceitos que representam construções comuns na modelagem do domínio. Estes padrões podem ser aplicáveis em um domínio ou em muitos
- Padrões de Arquitetura:
  - Descrevem a estrutura e o relacionamento dos componentes de um sistema de software
- Padrões de Processo
  - Descrevem modelos de processo

Programação Orientada a Objetos

Padrões de Projeto GoF (Gang of Four)

- Esses padrões tem como objetivo, solucionar problemas comuns de softwares que tenham algum envolvimento com orientação a objetos.
- São formados por três grupos:
  - **Padrões de criação**: **Factory Method**, Abstract Factory, **Singleton**, Builder, Prototype.
  - **Padrões estruturais**: Adapter, Bridge, **Composite**, Decorator, Facade, Flyweight, Proxy;
  - **Padrões comportamentais**: Chain of Responsability, Command, Interpreter, Iterator, Mediator, Memento, **Observer**, State, Strategy, Template, Method, Visitor.

Programação Orientada a Objetos

Características

- Algo comprovado - conhecimentos das melhores práticas
- Descoberto, não inventado
- Soluções Sucintas e de Fácil Aplicação
  - Capturam, de forma sucinta e facilmente aplicável, soluções de projeto de programas de computador que foram desenvolvidas e evoluíram com o passar do tempo

Programação Orientada a Objetos

Características

- Soluções Exaustivamente Refinadas
  - Resultados de um longo processo de projeto, reprojeto, teste e reflexão sobre o que torna um sistema mais flexível, reusável, modular e compreensível
- Soluções Compartilhadas
  - Construídas em grupo, através do uso de um vocabulário comum.

Documentação

- Nome:
  - Resume em uma ou duas palavras: o problema, as soluções e consequências do uso do padrão.
  - Deve ser facilmente lembrado, reflete o conteúdo do padrão.
- Problema:
  - Descreve quando aplicar o padrão.
  - Explica o problema e seu contexto. Sintomas e condições.

Documentação

- Solução:
  - Elementos que constituem o design, seus relacionamentos, responsabilidades e colaboradores.
- Consequências:
  - Resultados e compromissos decorrentes da aplicação do padrão.
  - Impactos sobre a flexibilidade, extensibilidade, portabilidade ou desempenho do sistema.

Programação Orientada a Objetos

Tipos de Padrões de Projeto

- **Padrões GoF de criação**
  - Exigem um tratamento de como os objetos são criados, para atenderem às diversas necessidades. No Java, os objetos são instanciados através de seus construtores, porém a utilização deles fica limitada quando:
- O código que referencia a criação de um objeto precisa conhecer os construtores dele, isso aumenta o acoplamento das classes.
- O objeto não pode ser criado parcialmente.
- O construtor não consegue controlar o número de instâncias presentes na aplicação.

Padrão de Criação Factory Method

- Nome:
  - **Factory Method**
- Problema:
  - Como adiar a decisão de que tipo de classe instanciar?
- Contexto:
  - Quando uma classe cria instâncias de outra classe, aumenta a dependência e o acoplamento entre elas, e é um grande problema para projetos. O ideal é tornar o acoplamento o menor possível.

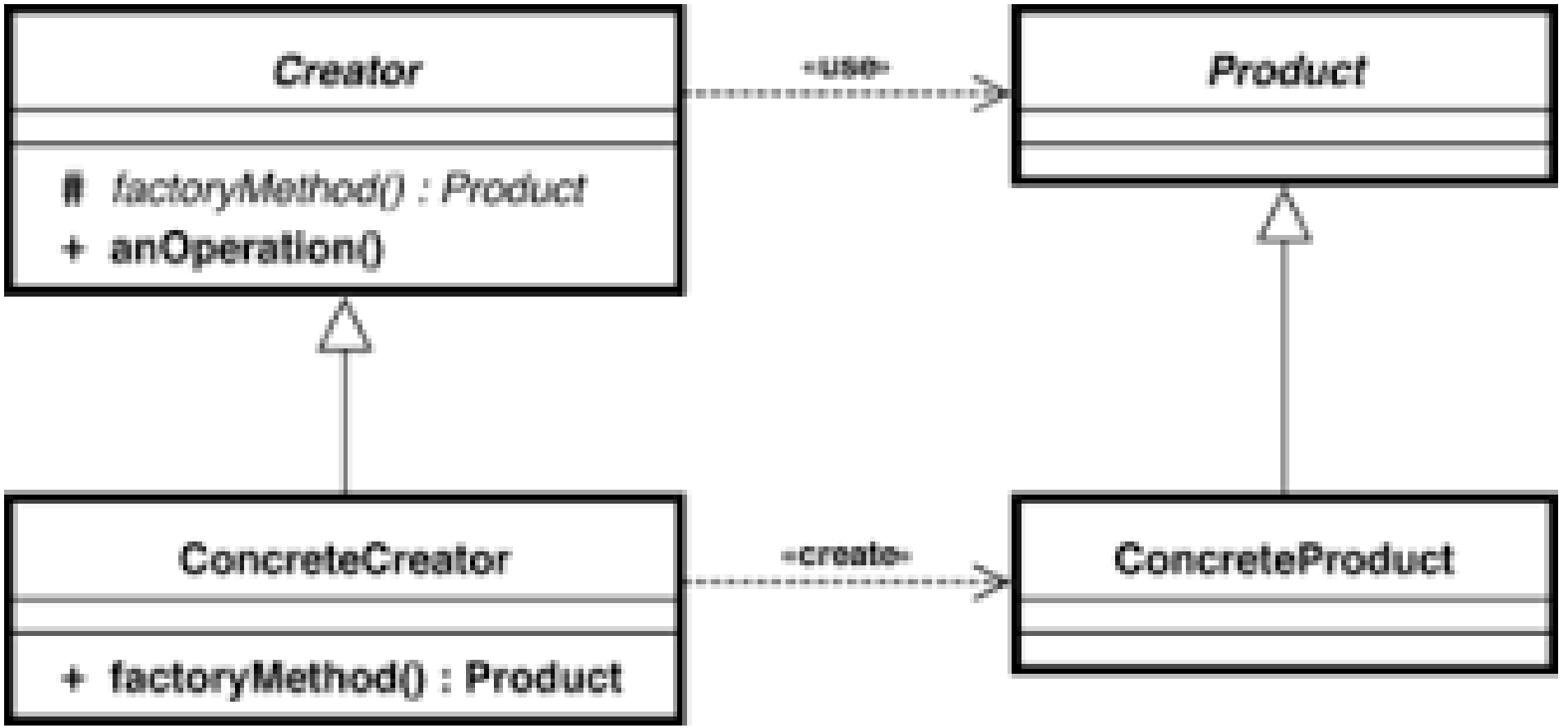
Padrão de Criação Factory Method

- Força:
  - Uso de classes abstratas e interfaces
  - Uso de uma fábrica de criação de objetos
  - Delegar a decisão de criação de objetos

Padrão de Criação Factory Method

- Solução:
  - Uma classe de criador abstrata que é a Creator que define um método fábrica abstrata que as subclasses implementam para criar um produto (factoryMethod) e pode possuir um ou mais métodos.
  - Cada ConcreteCreator implementará seu próprio método de criação.

Padrão de Criação Factory Method



Padrão de Criação Factory Method

- Vantagens:
  - encapsula-se a criação de objetos deixando as subclasses decidirem quais objetos criar – diminui o acoplamento
  - permite a uma classe decidir a instanciação para suas subclasses

Padrão de Criação Factory Method

–Exemplo de Uso:

```
public abstract class Fabrica {
      public abstract Pessoa criaPessoa(String nome, String sexo);
}
class FabricaPessoa extends Fabrica{
     public Pessoa criaPessoa(String nome, String sexo) {

        if (sexo.equals("M"))
            return new Homem(nome);
        else if (sexo.equals("F"))
                     return new Mulher(nome);
                else return new Mulher(null);
   }
}
```

Padrão de Criação Factory Method

- Exemplo de Uso:

```
public class TesteApp {
    public static void main(String args[]) {
        FabricaPessoa factory = new FabricaPessoa();
        String nome = "Carlos”;
        String sexo = “M”;
        Pessoa pessoa = factory.criaPessoa(nome, sexo);
    }
}
```

Padrão de Criação Factory Method

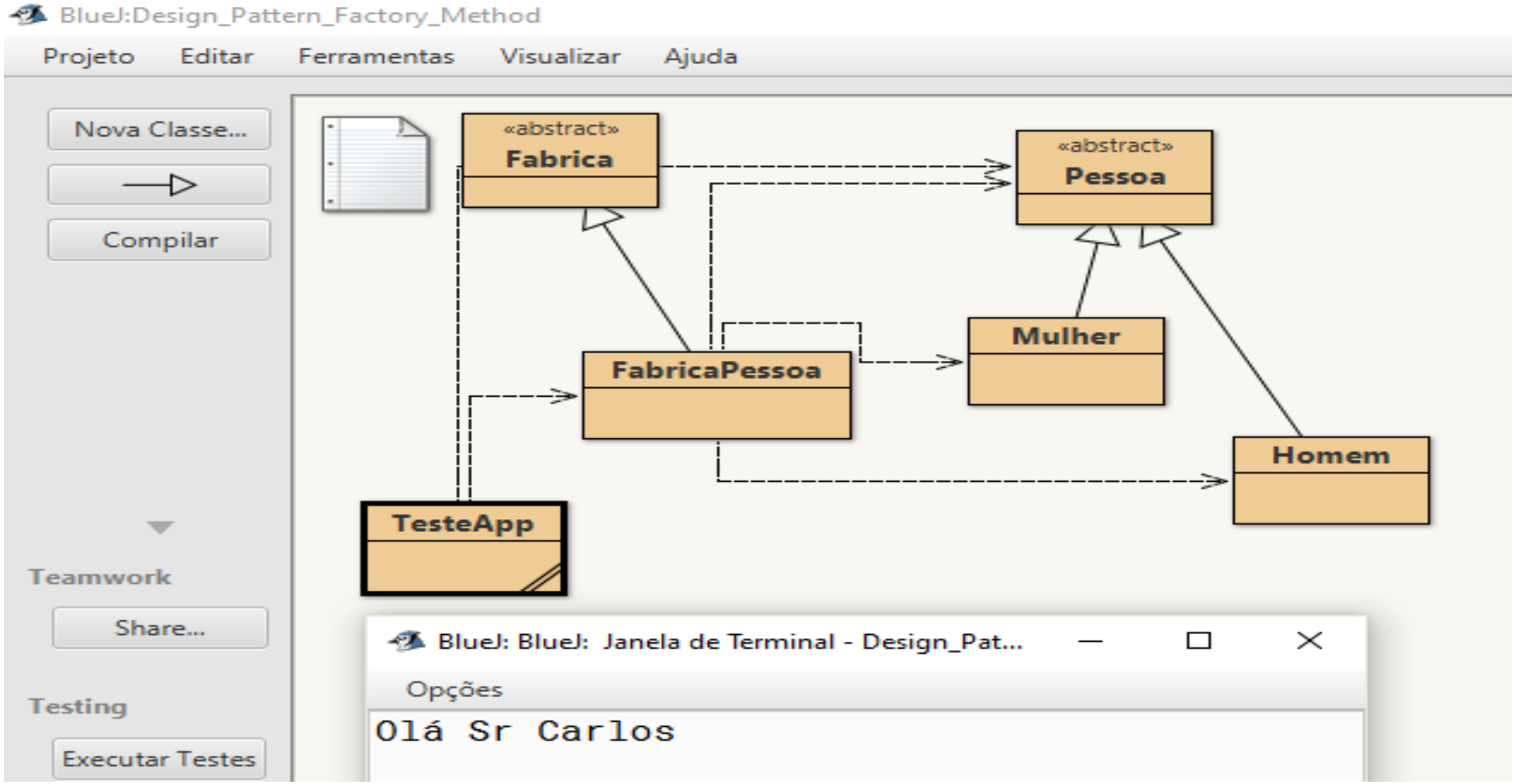
- Exemplo de Uso:

```
public abstract class Pessoa {
    public String nome;
    public String sexo;
}
class Homem extends Pessoa {
    public Homem(String nome) {
        this.nome = nome;
        System.out.println("Olá Sr " + nome);
    }
}
class Mulher extends Pessoa {
    public Mulher(String nome) {
        this.nome = nome;
        System.out.println("Olá Sra " +nome);
    }
 }
```

Programação Orientada a Objetos

Padrão de Criação Factory Method

– Rodando no BlueJ



Padrão de Criação **Singleton**

- Nome:
  - **Singleton**
- Problema:
  - Como pode ser construída uma classe que só pode ter uma única instância e que pode ser acessada globalmente dentro da aplicação?
- Contexto:
  - Em algumas aplicações, uma classe deve ter exatamente uma instância.
- Força:
  - Criar variáveis globais, usar variáveis de classe (estáticas)

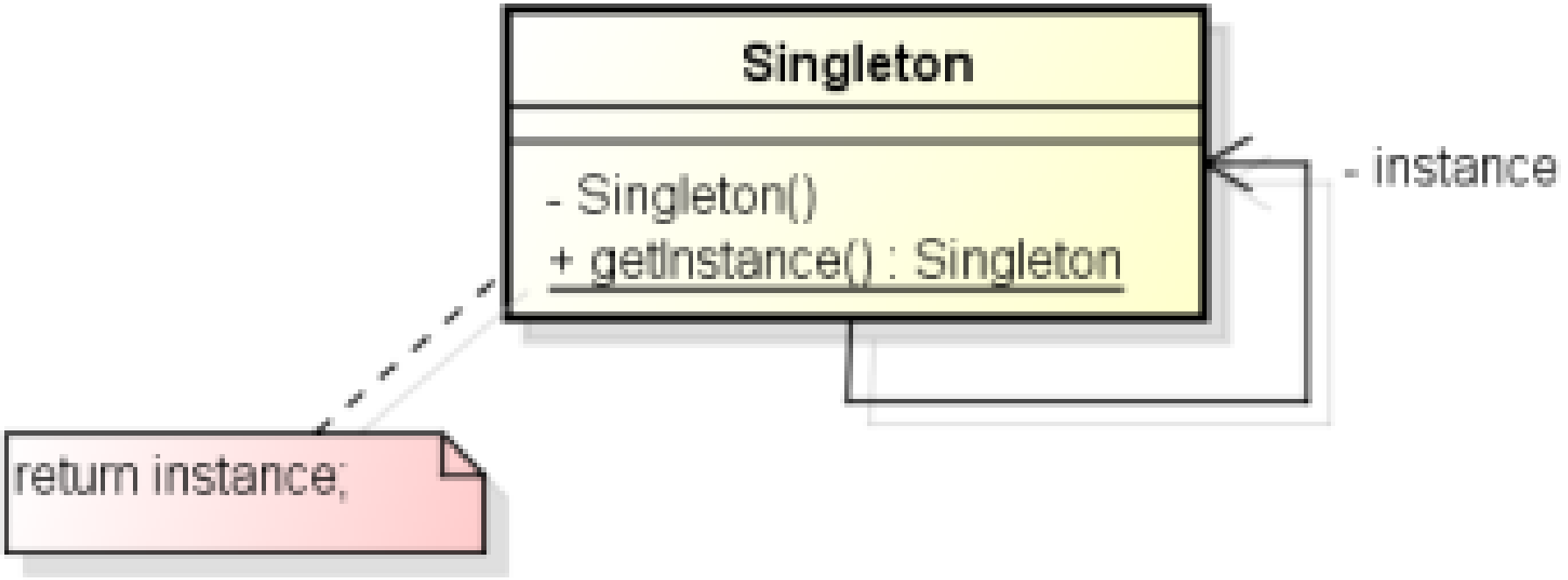
Padrão de Criação **Singleton**

- Solução:
  - Criar uma classe com um método de classe (ou estática) getInstance(), que:
- Quando a classe é acessada pela primeira vez, a instância do objeto é criada e retornada para o cliente.
- Nos acessos subsequentes ao método getInstance(), nenhuma instância adicional é criada, mas a identidade do objeto existente é retornada.

Padrão de Criação **Singleton**

- Alguns aspectos que devem serem observados ao criar esse padrão:
  - O construtor da classe fica como privado (private), sendo que não pode ser instanciada para fora da própria classe.
  - A classe é final, pois não permite a criação de subclasses da própria classe.
  - O acesso é permitido através do método que retorna a instância única da classe, ou faz a criação de uma, caso não tenha sido criada.

Padrão de Criação **Singleton**



Padrão de Criação **Singleton**

- Vantagens:
  - Oferece acesso controlado à única instância do objeto, pois a classe Singleton encapsula a instância.
  - Uma variação do padrão pode ser usada para criar um número específico de instâncias, se for requerido.

Padrão de Criação **Singleton**

- Exemplo de Uso:
  - Uma aplicação bancária onde cada conta tem um vínculo com um banco.
  - O banco deve ser único para todas as contas.

```
public final class Banco {
        public static Banco instancia;
        int codigo;
        String nome;
        private Banco() {
                codigo = 001;
                nome = "Banco do Brasil";
        }
        public static Banco getInstancia() {
                if (instancia == null) {
                        instancia = new Banco();
                }
                return instancia;
        }
}
```

Padrão de Criação **Singleton**

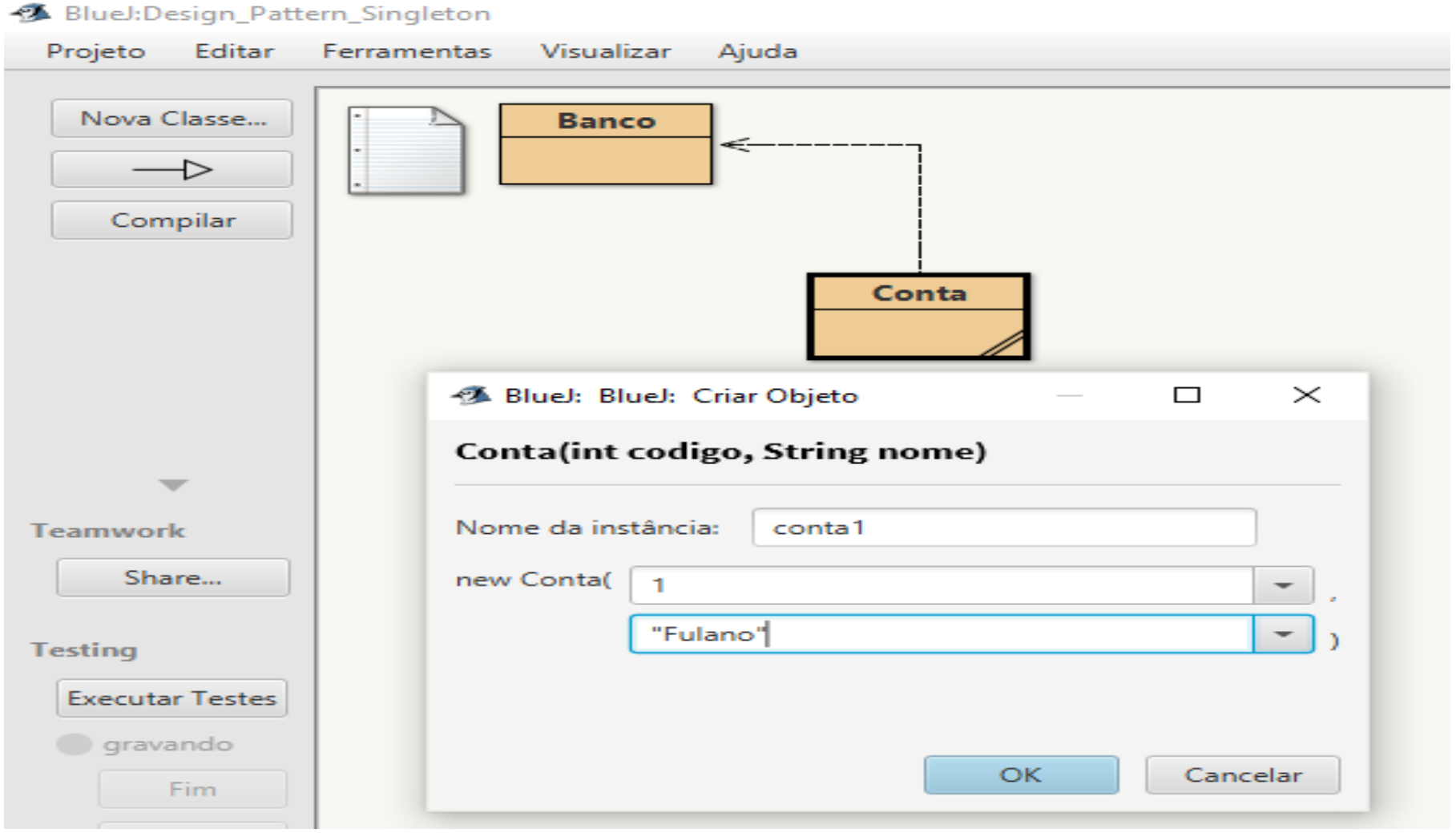
- Exemplo de Uso:
  - Uma aplicação bancária onde cada conta tem um vínculo com um banco.
  - O banco deve ser único para todas as contas.

```
public class Conta {

        Banco banco;
        int codigo;
        String nome;

        Conta(int codigo, String nome) {
                banco = Banco.getInstancia();
                this.codigo = codigo;
                this.nome = nome;

        }
}
```

Programação Orientada a Objetos

Padrão de Criação **Singleton**

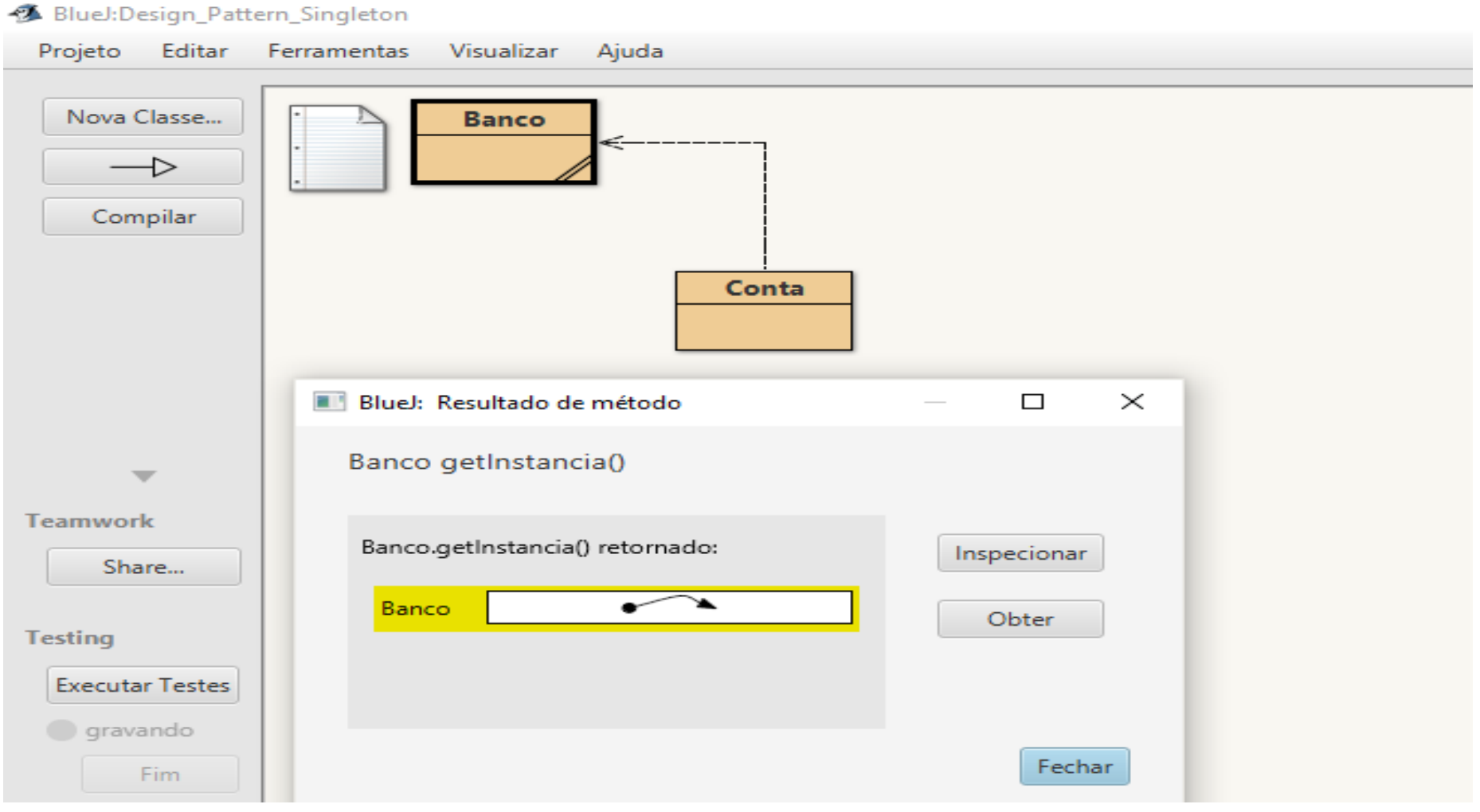
- Rodando no BlueJ



Programação Orientada a Objetos

Padrão de Criação **Singleton**

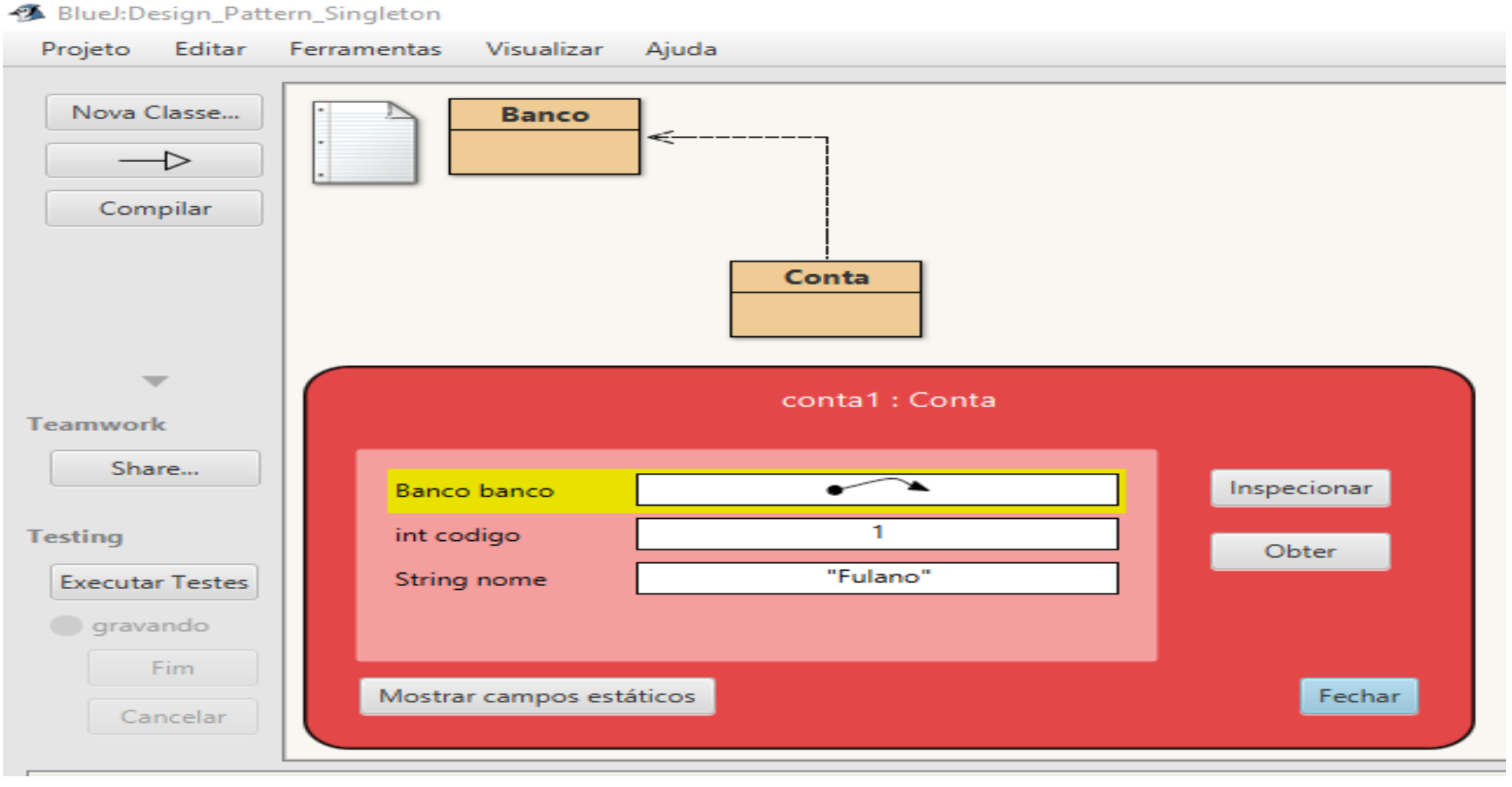
- Rodando no BlueJ



Programação Orientada a Objetos

Padrão de Criação **Singleton**

- Rodando no BlueJ



Tipos de Padrões de Projeto

- Padrões GoF **Estruturais**
  - Esse padrões descrevem os seguintes aspectos: elaboração, associação e a organização entre objetos e classes/interfaces. Permitem combinar objetos em estruturas mais complexas, ou descrever como as classes são herdadas ou compostas a partir de outras.

Padrão Estrutural Composite

- Nome:
  - **Composite**
- Problema:
  - Se existe um requisito para representar hierarquias todo-parte, então ambos objetos (todo e parte) oferecem a mesma interface para objetos cliente.
- Contexto:
  - Numa aplicação tanto o objeto que contém quanto o que é componente devem oferecer o mesmo comportamento.
  - Objetos cliente devem ser capazes de tratar objetos compostos ou componentes do mesmo jeito.

Padrão Estrutural Composite

- Forças:
  - O requisito que os objetos, tanto composto como componente, ofereçam a mesma interface, sugere que eles pertençam à mesma hierarquia de herança.
  - Isto permite que operações sejam herdadas e redefinidas com a mesma assinatura (polimorfismo).
  - A necessidade de representar hierarquias todo-parte indica a necessidade de uma estrutura de agregação

Padrão Estrutural **Composite**

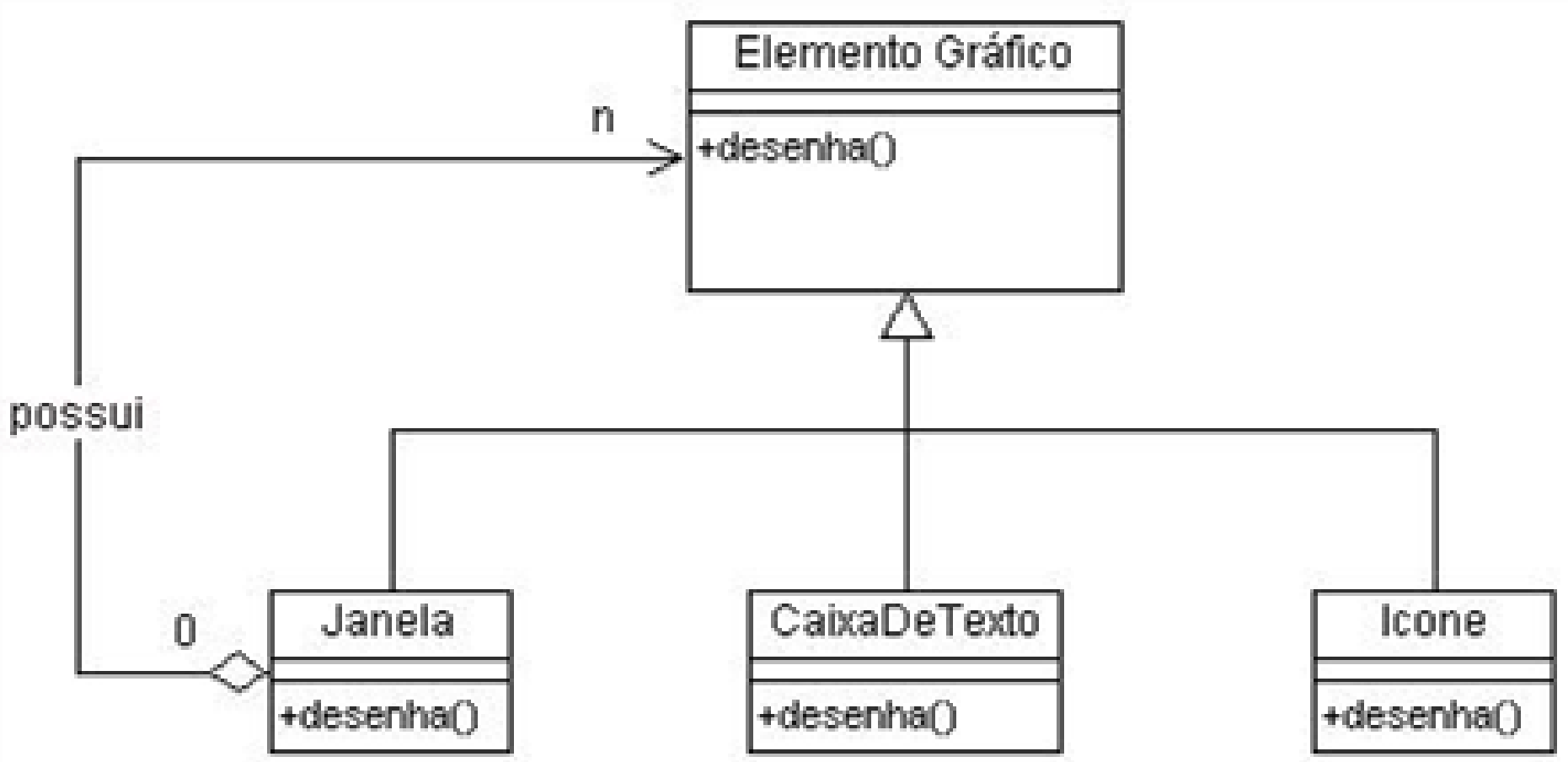
- Solução:
  - Combinar hierarquias de herança e agregação
  - Ambas subclasses, Leaf e Composite, têm uma operação redefinida usando polimorfismo anOperation().

Padrão Estrutural Composite

- Solução:
  - Na classe *Composite* esta operação redefinida invoca a operação relevante a partir de seus componentes usando um *loop*.
  - A subclasse *Composite* também tem operações adicionais para gerenciar a hierarquia de agregação, então os componentes podem ser adicionados ou removidos.

Padrão Estrutural **Composite**

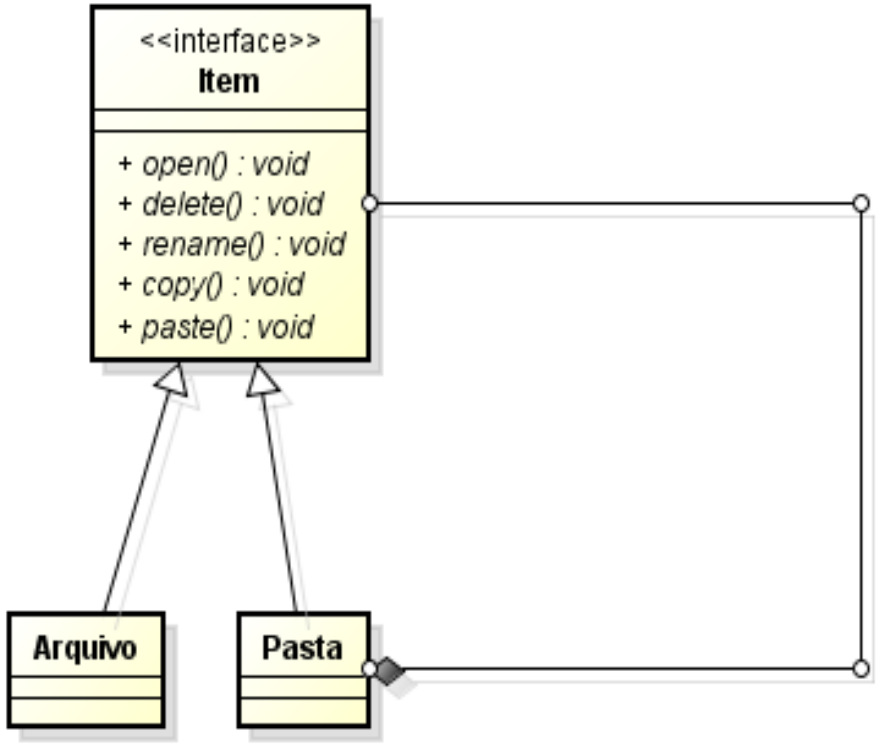
- Exemplo : Telas

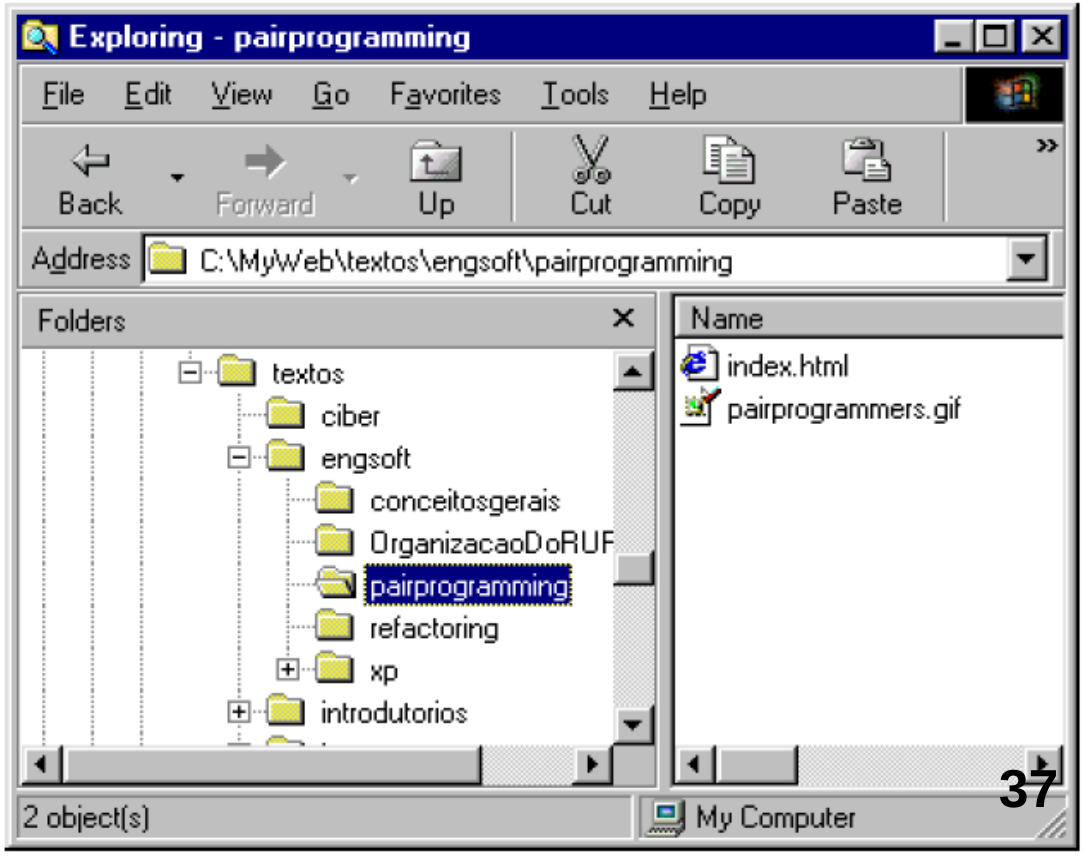


Programação Orientada a Objetos

Padrão Estrutural **Composite**

- Exemplo : Sistema de Arquivos

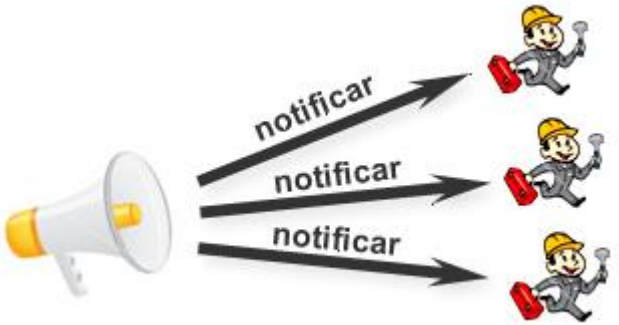




Tipos de Padrões de Projeto

- Padrões GoF Comportamentais
  - Esses padrões, mostram o processo de como os objetos ou classes se comunicam. Em geral, buscam um **baixo acoplamento** entre os objetos, apesar da comunicação que existe entre eles.

Padrão Comportamental **Observer**

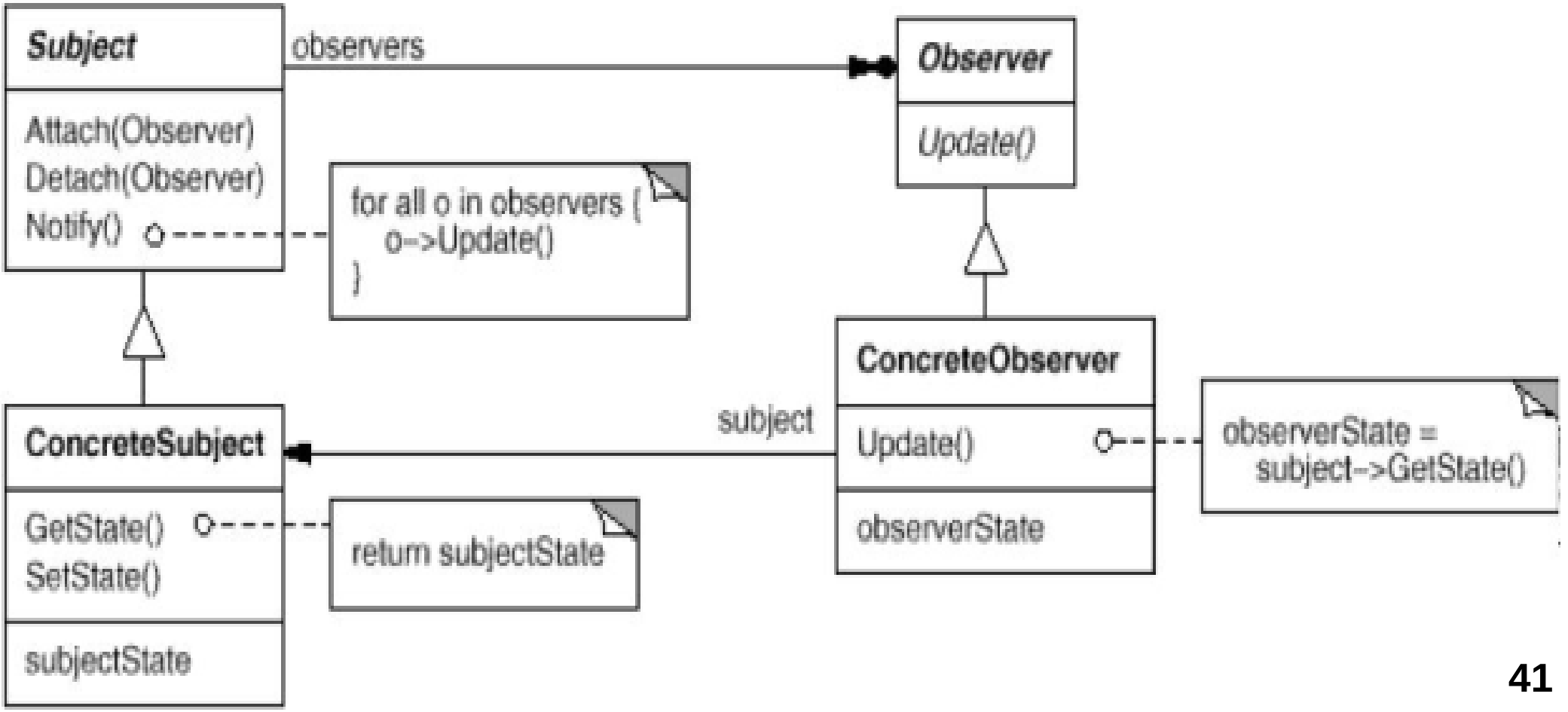


- Nome:
  - **Observer**
- Problema:
  - Os dados do componente sujeito modificam-se constantemente e precisam ser atualizados nos outros componentes. O número de componentes pode variar.
- Contexto:
  - Situações nas quais vários componentes dependem de dados que são modificados em outro componente (sujeito).

Padrão Comportamental **Observer**

- Solução:
  - Utilizar um mecanismo de registro que permite ao componente sujeito notificar aos interessados sobre mudanças.
  - Exemplos: atualizações de software, facebook, pacote swing (listeners), etc.

Padrão Comportamental Observer



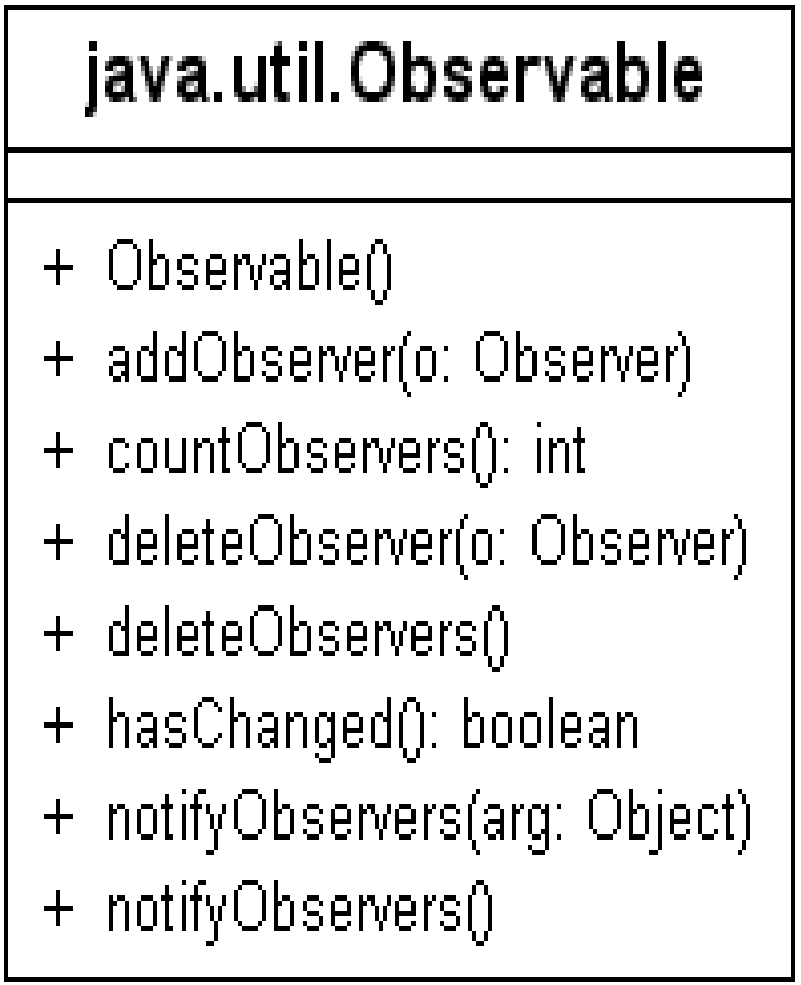
Padrão Comportamental **Observer**

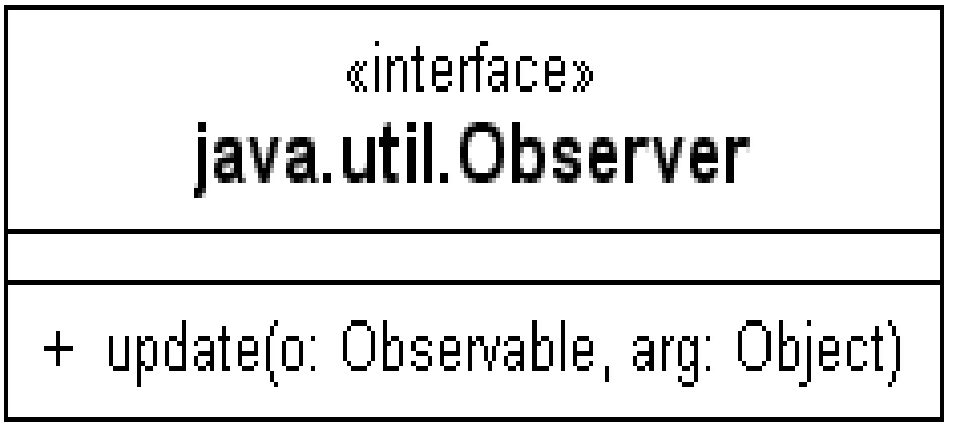
- **Interface pública Observer**:
  - As classes que implementarem essa interface, que por sua vez conterá um método de atualização (update), deverão ser notificadas quando da atualização ou mudança do estado do objeto observado.

Padrão Comportamental *Observer*

- **Classe Observable**: Define a classe que será observada.
  - Essa classe geralmente guarda como atributo uma lista dos *observers* associados à mesma, para que os mesmos possam ser notificados quando a mudança acontecer na classe.

Programação Orientada a Objetos





Exemplos de uso:

- O padrão Observer é bastante comum no código Java, especialmente nos componentes da interface de usuário.
- Ele fornece uma maneira de reagir a eventos que acontecem em outros objetos sem acoplamento às suas classes.

Programação Orientada a Objetos

Aqui estão alguns exemplos do padrão nas principais bibliotecas Java:

- **java.util.Observer**/**java.util.Observable** (raramente usado com o surgimento de uma nova biblioteca inspirada no padrão)
- Todas as implementações de **java.util.EventListener** (praticamente em todos os componentes Swing)
- **javax.servlet.http.HttpSessionBindingListener**
- **javax.servlet.http.HttpSessionAttributeListener**
- **javax.faces.event.PhaseListener**

Identificação

- O padrão pode ser reconhecido por métodos de assinatura, que armazenam objetos em uma lista e por chamadas para o método de atualização emitido para objetos nessa lista.

Programação Orientada a Objetos

- Questões Relacionadas ao uso de Padrões:
  - Existe algum padrão que trata um problema similar?
  - O contexto do padrão é consistente com o problema real?
  - O padrão tem uma solução alternativa que pode ser mais aceitável?
  - Existe uma solução mais simples?
  - Existem algumas restrições que sejam impostas pelo ambiente de software que está sendo usado?

Programação Orientada a Objetos

Exercício:

1. Copie o código abaixo, execute e veja o resultado.

```
class Numeracao {
   private static int seq = 0;
   private int numero;

   public Numeracao() {
      numero = ++seq;
   }

   public String toString() {
      return "Numeração " + numero;
   }
}
```

```
public class TesteNumeracao {
   public static void main(String[] args) {
      for (int i = 0; i < 10; i++) {
         Numeracao num = new Numeracao();
        System.out.println(num);
      }
   }
}
```

Programação Orientada a Objetos

Rodando o exercício no BlueJ: